CSRD (2009) 24: 21–31
DOI 10.1007/s00450-009-0089-1

SPECIAL ISSUE PAPER

A novel multiple-walk parallel algorithm for the Barnes–Hut treecode on GPUs – towards cost effective, high performance N-body simulation

Tsuyoshi Hamada · Keigo Nitadori · Khaled Benkrid · Yousuke Ohno · Gentaro Morimoto · Tomonari Masada · Yuichiro Shibata · Kiyoshi Oguri · Makoto Taiji

Published online: 20 May 2009

Abstract Recently, general-purpose computation on graphics processing units (GPGPU) has become an increasingly popular field of study as graphics processing units (GPUs) continue to be proposed as high performance and relatively low cost implementation platforms for scientific computing applications. Among these applications figure astrophysical N-body simulations, which form one of the most challenging problems in computational science. However, in most reported studies, a simple $\mathcal{O}(N^2)$ algorithm was used for GPGPUs, and the resulting performances were not observed to be better than those of conventional CPUs that were based on more optimized $\mathcal{O}(N \log N)$ algorithms such as the tree algorithm or the particle-particle particle-mesh algorithm. Because of the difficulty in getting efficient implementations of such algorithms on GPUs, a GPU cluster had no practical advantage over general-purpose PC clusters for N-body simulations. In this paper, we report a new method for efficient parallel implementation of the tree algorithm on GPUs. Our novel tree code allows the realization of an N-body simulation on a GPU cluster at a much higher performance than that on general PC clusters. We practically performed a cosmological simulation with 562 million particles on a GPU cluster using 128 NVIDIA GeForce 8800GTS GPUs at an overall cost of 168 172 $. We obtained a sustained performance of 20.1 Tflops, which when normalized against a general-purpose CPU implementation leads to a performance of 8.50 Tflops. The achieved cost/performance was hence a mere $19.8 /Gflops which shows the high competitiveness of GPGPUs.

Keywords GPU · N-body simulation · Tree algorithm

T. Hamada (✉) · T. Masada · Y. Shibata · K. Oguri
Faculty of Engineering, Department of Computer and Information Sciences, Nagasaki University,
Bunkyo-machi,
852-8521 Nagasaki, Japan
e-mail: hamada@cis.nagasaki-u.ac.jp
e-mail: masada@cis.nagasaki-u.ac.jp
e-mail: shibata@cis.nagasaki-u.ac.jp
e-mail: oguri@cis.nagasaki-u.ac.jp

K. Nitadori
Department of Astronomy, University of Tokyo,
7-3-1 Hongo, Bunkyo-ku,
113-0033 Tokyo, Japan
e-mail: nitadori@cfca.jp

K. Benkrid
School of Engineering, The University of Edinburgh,
King's Buildings, Mayfield Road,
Edinburgh EH9 3JL, UK
e-mail: k.benkdird@ed.ac.uk

Y. Ohno · G. Morimoto · M. Taiji
RIKEN (The Institute of Physical and Chemical Research),
61-1 Ono, Tsurumi, Yokohama,
230-0046 Kanagawa, Japan
e-mail: ohno@riken.jp
e-mail: gentaro.morimoto@riken.jp
e-mail: taiji@riken.jp

1 Introduction

Astronomical N-body simulations have been widely used to investigate the formation and evolution of various astronomical systems such as planetary systems, globular clusters, galaxies, clusters of galaxies, and other such large scale structures in the universe. In such simulations, planetesimals, stars, or galaxies are considered to be particles that interact with each other through Newtonian gravity. The interactions between the particles are numerically evaluated and the particle positions are updated according to Newton's equation of motion.

In many cases, the size of an astrophysical N-body simulation is limited by the availability of computational resources. Since gravitational interactions are long-range interactions, the calculation cost of the interactions between particles in the simplest scheme is $\mathcal{O}(N^2)$ per timestep, where N is the number of particles in the system. This cost can be further reduced to $\mathcal{O}(N \log N)$ by using approximation algorithms such as the Barnes–Hut tree algorithm [1]; however, the scaling coefficient of this algorithm is considerably large. Thus, the calculation cost of the interactions between particles is usually the most expensive part of a calculation and limits the number of particles that can be handled by a simulator.

The gravitational force exerted on a particle (particle i) by other particles (particle j) can be given as:

$$\boldsymbol{a}_i = \sum_j \frac{m_j \boldsymbol{r}_{ij}}{r_{ij}^3}, \tag{1}$$

where $\boldsymbol{a}_i$ is the gravitational acceleration of the i-th particle (hereafter, we will refer to it as the i-particle), $\boldsymbol{r}_j$ and m_j are the position and mass, respectively, of the j-th particle (hereafter referred to as the j-particle), and $\boldsymbol{r}_{ij} = \boldsymbol{r}_j - \boldsymbol{r}_i$.

Astrophysical N-body simulations have been one of the most challenging problems in computational science. Indeed, in the years 1992, 96, 97, 98, and 99, Gordon Bell prizes were awarded to studies on cosmological N-body simulations [2–5]. Gordon Bell prizes were also awarded in 1995, 2000, and 2001 to studies on N-body simulations of a black hole binary in a galaxy [6–8] and to a study on the simulation of a proto-planetary system in 2003 [9].

For cosmological N-body simulations, the tree algorithm [1] has been widely used in several studies, including in the above-mentioned Gordon Bell winners [2–5, 10]. In this algorithm, the particles are organized in the form of a tree, and each node of the tree represents a group of particles. The forces from the particles in a distant node are replaced by the force from their center of mass. Recently, very large-scale cosmological simulations (up to 10 billion particles) using a tree algorithm running on a massive parallel supercomputer (IBM Regatta [11]) and a Beowulf-type PC-cluster[3, 12] have been reported.

In this paper, we report a novel approach to the tree algorithm and evaluate the performance of the algorithm for astrophysical N-body simulations on a cluster of NVIDIA GeForce 8800GTS GPUs. The GPUs in this study are used for calculating the gravitational forces. They are connected to an x86-64 PC cluster and operate as hardware accelerators that calculate gravitational forces. Other operations such as tree construction, tree traversal, and time integration are performed on the host computers.

Recently, techniques to increase the speed of N-body simulations on GPUs have become an active area of research. Nyland et al. reported a GPU performance of 11 Gflops for a 4096-body simulation in [13], while Harris reported performances of 18 Gflops and 26 Gflops for a 8192-body simulation in [14, 15]. Zwart [16] reported a GPU implementation capable of 30 Gflops on an NVIDIA GeForce 8800GTX GPU. A performance of 256 Gflops for a 131 072-body simulation on an NVIDIA GeForce 8800GTX was reported by Hamada et al. [17]. Nyland et al. reported a performance of 342 Gflops for a 3076-body simulation on an NVIDIA GeForce 8800GTX [18]. Belleman et al. also reported a GPU performance of 340 Gflops for a 131 072-body simulation on an NVIDIA GeForce 8800GTX [19]. In addition, a study conducted by Hamada reported a performance of 653 Gflops for a 131 072-body simulation on an NVIDIA GeForce 8800GTS [20][1].

These studies measured the performance while considering algorithms that used the $\mathcal{O}(N^2)$ direct summation method and a shared timestep scheme, where all the particles are integrated within a single global timestep. Such a simple, brute-force, algorithm can be used to achieve a high flops count but is of little practical importance. On the other hand, the tree algorithm, which has a calculation cost of $\mathcal{O}(N \log N)$, is widely used in practical scientific simulations. However, the standard tree algorithm has so far resulted in very low efficiency implementations when executed on GPUs.

In order to parallelize the tree algorithm, we need to replace the i-particles in (1) into groups of particles which shares the same list of j-particles[21]. The group of i-particles that share the same interaction list as that of j-particles is known as a "walk." Since i-particles in the same walk should be located close to each other, the size of a walk cannot be as large as that in case of the brute-force all-to-all summation scheme used in most kernel benchmarks. The typical number of i-particles in a walk is several hundred, at most, in the tree algorithm. Until now, no one has succeeded in developing an algorithm in which thousands of GPU threads can calculate the forces on several i-particles simultaneously with high efficiency [22].

In this paper, we report a breakthrough approach to the tree algorithm on GPUs. One of the most commonly occurring problems in previous tree algorithms executed on GPUs was the formation of bottlenecks; this was due to the inefficient use of parallel pipelines when the number of i-particles in a walk was small and because each walk was evaluated separately. In order to solve this problem and achieve high efficiency, we developed a method that could pack multiple walks that are then evaluated by the GPU simultaneously. In order to implement this novel approach, we redesigned the tree algorithm. As a result, a drastic gain in performance was achieved as compared to previous implementations of the tree algorithm on GPUs.

[1] http://progrape.jp/cs/.

With our approach, the tree algorithm shows a significant performance gain when executed on GPUs as compared to the performances of the conventional tree algorithm running on conventional microprocessors. The previous implementations of the tree algorithm made it difficult to achieve an effective GPU performance, especially compared to the performances of conventional PC clusters. Using our novel approach, however, a GPU cluster can outperform a PC cluster from the viewpoints of cost/performance, power/performance, and physical dimensions/performance. In order to test our implementation method, we developed a large-scale GPU cluster with 128 GPUs and investigated the results of the application of our method to real scientific problems. We performed large-scale cosmological simulations of 562 million particles using the tree algorithm on our GPU cluster, and we obtained a sustained effective performance of 8.5 Tflops, which corresponds to a cost performance of a mere \$19.8 /Gflops.

The remainder of this paper is organized as follows. In Sect. 2, we present the parallel tree algorithm and discuss the problems of the previous approaches to implementing it. Section 3 then presents our novel GPU implementation of the tree algorithm. It first presents the hardware configuration of our GPU cluster before it gives real implementation results of a cosmological N-body simulation engine that uses our novel tree algorithm implementation approach on the GPU cluster. Finally, conclusions are drawn.

2 Parallel tree algorithm

The parallel code [23] that we have developed is based on Barnes' modified tree algorithm [21]. The calculation cost of this algorithm is much less than that of the conventional algorithm. In our code, the forces exerted by multiple particles are calculated in parallel using accelerator hardware such as vector processors. In the original algorithm, an interaction list is created for each particle. In the modified tree algorithm, the neighboring particles are grouped together and the particles in the same group share an interaction list. It is possible to directly calculate the forces exerted by the particles in the same group.

The modified tree algorithm reduces the calculation cost roughly by a factor of n_g, where n_g is the average number of particles in the group. On the other hand, the number of operations increases as we increase n_g, since the interaction list becomes longer. There is, therefore, an optimal value of n_g for which the total computation time is minimum. The optimal value of n_g strongly depends on the ratio of the speed of the host PC to that of the GPUs. For the configuration used in this paper, the optimal n_g is around 500.

Parallelization was performed by space decomposition using an orthogonal multisection method, which is a generalization of the orthogonal recursive bisection (ORB) method [23]. We used the special division of an $8 \times 4 \times 4$ grid. The advantage of the present method is that the number of processors is not required to be a power of two. We used the local essential tree scheme (LET scheme) [23] for parallelization. Other schemes such as the hashed-oct-tree (HOT)[10] method can also be used to parallelize the tree algorithm. The reason we selected the LET scheme is because its communication pattern is much simpler than that of the others.

2.1 Previous GPU implementations for N-body force calculation and performance in the tree algorithm

In this subsection, we discuss the reason why previous N-body simulations implemented on GPGPUs could not outperform equivalent implementations on CPUs, even though the kernel showed high efficiency when executed on GPUs.

We compared several different approaches to the N-body simulation algorithm using the same hardware. We compared (1) CUNBODY-1.0 by Hamada et al. (Hereafter Hamada 2007) [17], (2) Kirin by Belleman et al. (Belleman 2007) [19], (3) the algorithm proposed by Nyland et al. for the GPU Gems 3 (Nyland 2007) [18], and (4) the modified algorithm proposed in this work. Since we do not have the original source code for Kirin's implementation, the corresponding results are measured through our own implementation of their method as described in their paper. We also considered the results of the tree algorithm executed on a conventional processor with the highly-optimized code "Phantom-GRAPE (GRAvity PipE)" developed by Nitadori et al. [24]. This code was deeply optimized using Intel's streaming SIMD extensions (SSE) in the assembly language; this code is known as one of the fastest variations of the tree algorithm for general-purpose processors.

Figure 1 shows the resulting performances of the N-body kernel using both the $\mathcal{O}(N^2)$ direct-summation, and the kernel that uses the $O(N \log N)$ tree algorithm with the shared time-step scheme. We used a single NVIDIA GeForce 8800GTS GPU connected to an Intel Core 2 Quad Q6600 CPU based PC. The Phantom-GRAPE was executed only on the CPU. Plummer spheres with 65 536 and 4 194 304 particles were used for the direct algorithm and the tree algorithm, respectively. We assumed the softening parameter $\varepsilon = 0.025$.

In the case of the $\mathcal{O}(N^2)$ scheme, all of the four variations of the algorithm executed on the GPU showed a better performance than the one executed on the CPU (Phantom-GRAPE). These results were consistent with those of previous reports. The differences in the performances of the four implementations are as follows. The approaches of Belleman 2007 and Nyland 2007 are essentially the same, except

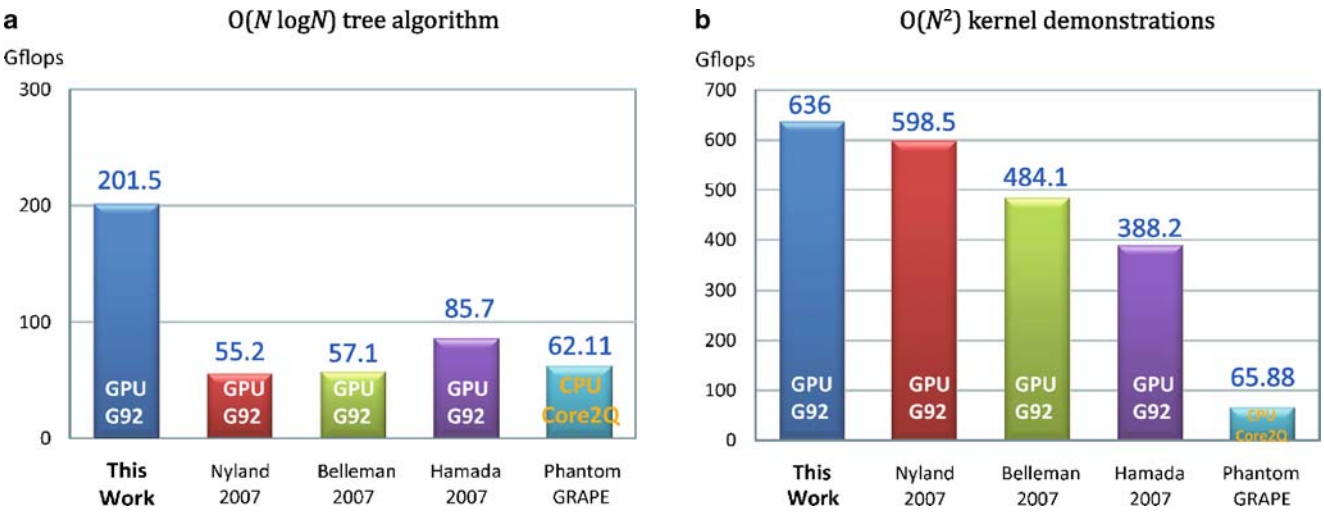

Fig. 1 Comparison of the performances of the N-body kernel using (**a**) the $\mathcal{O}(N^2)$ direct-summation approach and (**b**) the $\mathcal{O}(N \log N)$ tree algorithm with shared timestep scheme

for the number of threads. Since the approach by Nyland 2007 uses a larger number of threads (32 768 instead of 2048), their implementation exhibited a better performance. The performance of CUNBOODY-1.0/Hamada 2007 was poor due to an overhead of the force reduction since it was designed to operate efficiently with the tree algorithm. This problem and the details of our proposed approach will be explained later.

In the case of tree algorithms, except in the case of our new approach, the performance gains by the GPU were almost negligible (Belleman 2007 and Nyland 2007) or were fairly small (Hamada 2007) compared to CPUs. In order to explain this performance degradation, we first describe the main concepts of the past implementations.

There were two approaches in the previous implementations. Figure 2 shows the first approach taken by Nyland et al. [18] and Belleman et al. [19]. Each row corresponds to a processor in the GPU, and each group of rows corresponds to a SIMD block. The horizontal axis indicates the time of processing, and the blue circles indicate threads. In this case,

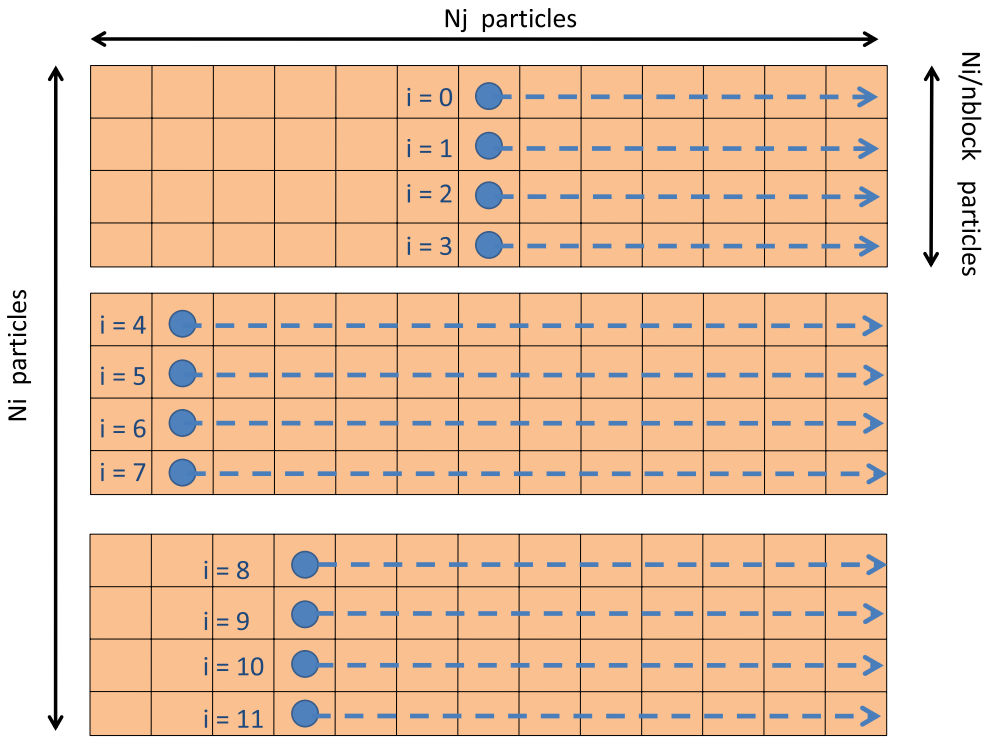

Fig. 2 Implementation of the GPU code reported by Nyland et al. [18] and Belleman et al. [19]

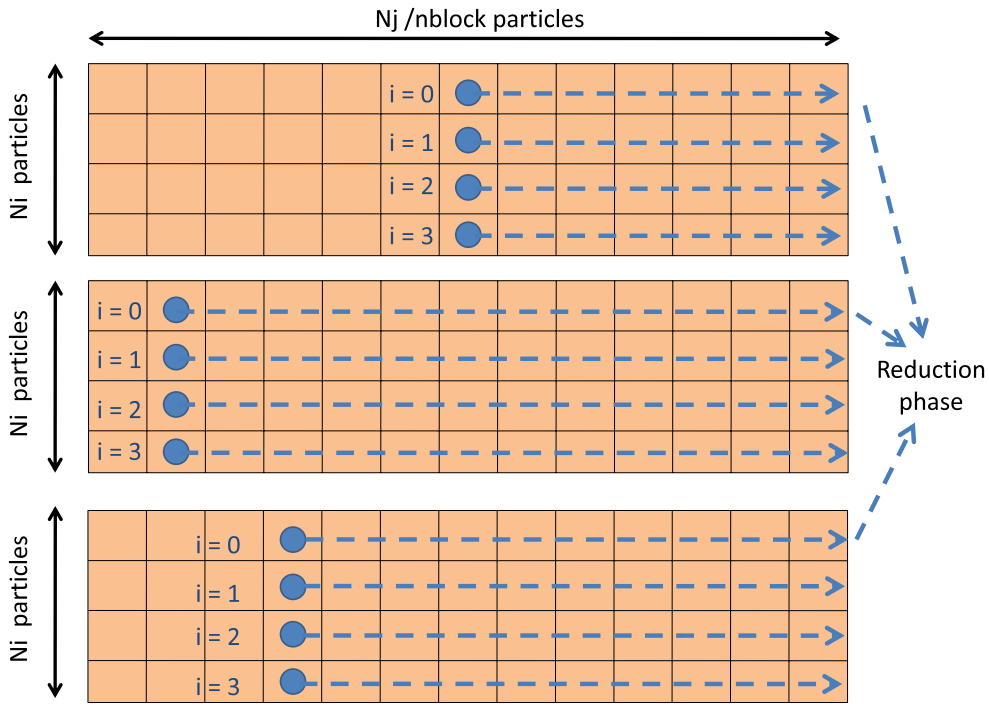

Fig. 3 Implementation of GPU code reported by Hamada et al. [22]

each thread calculates the force on a different i-particle – we call this the "i-parallel" approach. In this method, when the number of i-particles in a walk n_g (which shares the same interaction list as the j-particles) is smaller than the number of physical processor units, the rest of the processors are kept idle. Since n_g is typically of the order of several hundreds, the efficiency of processor usage tends to be low in this implementation.

The second, improved approach proposed by Hamada et al. [22] is shown in Fig. 3. In this case, the so-called "j-parallel" approach is used in addition to the i-parallel approach. The j-particles are divided into several groups, and the partial forces on the i-particles are calculated by different blocks. Multiple threads on each block evaluate different i-particles. The number of parallel i-particles is equal to the number of physical processors divided by the number of blocks, N_{blocks}, which is usually smaller than ng. Thus, the performance is better than that of the first approach, contrary to the results of the kernel benchmarks (see Fig. 1 where considerable degradation in the overall performance is revealed, and the increase in gain compared to that of the Phantom-GRAPE is rather small). This is due to the overhead of the reduction operations required for the partial forces. Indeed, in this approach, partial forces on an i-particle are calculated by different blocks; thus, they have to be summed up. Unfortunately, the reduction operations on a GPU are slow because of the large overhead involved in thread synchronization. There is a report available on the reduction operations that run on GPUs [25]; however, it is only valid for a reduction of large arrays with thousands of elements. Therefore, currently, the fastest method of reduction is to use the host CPU. Thus, the amount of communication between the GPU and the host CPU increases by N_{blocks}. Since the CPU-GPU communication is rather slow compared to the GPU speed, even with PCI Express Generation 2, the performance of the algorithm is greatly affected. Moreover, such a small gain in the CPU implementation will not prove the validity of a GPU cluster for real applications.

2.2 Proposed novel approach

In this section, we explain the details of our novel approach and present its advantages over previous approaches. The remaining problems and their possible solutions will also be discussed.

Figure 4 shows our new effective implementation of the tree algorithm on GPUs. Here, multiple walks are evaluated by different GPU blocks in parallel. In all previous implementations, only a single walk was calculated at a time. In the figure, the operations of three blocks are depicted. The threads in each block evaluate different i-particles of the same walk. The sequence of the algorithm is as follows:

1. First, the data of the multiple walks are prepared, i.e., the lists of i-particles and j-particles of each walk are prepared. The number of walks in a calculation is determined by the resource limitations of a GPU and its efficiency.
2. Multiple walks are then sent to a GPU.

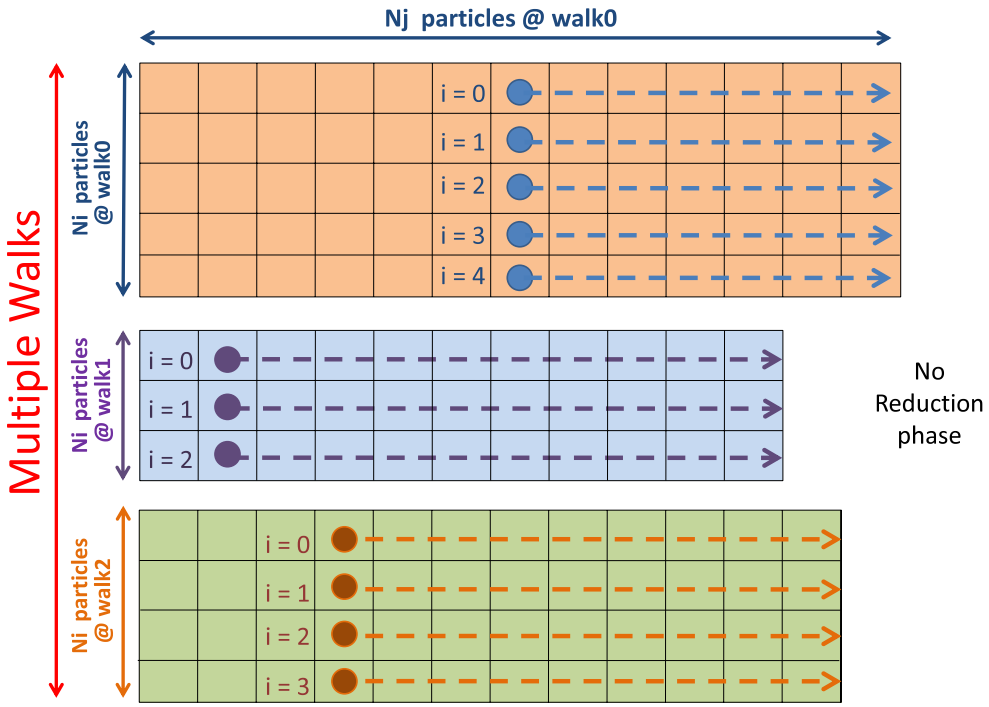

Fig. 4 Implementation of the novel GPU code reported by this paper

3. Calculations are performed. The GPU is partitioned so that each GPU block evaluates only a single walk.
4. The forces calculated by the different blocks in the GPU are received in a bundle.
5. The orbits and other calculations are integrated.
6. The process is repeated.

Our approach has several advantages. First, the reduction of partial forces is no longer necessary since each walk is calculated in a block and no j-parallelization is used. This solves the problem in the previous approach proposed by Hamada et al. Second, multiple walks are sent to a GPU simultaneously, and the forces of the multiple walks are also retrieved from the GPU at the same time. This enables more efficient communication between the CPU and GPU, since the number of DMA requests decreases, and the length of each DMA transfer becomes longer. Third, it makes parallelization in the host computation easier. Our approach involves the calculation of multiple walks at the same time; therefore, each thread in a host can process a group of different walks in parallel.

Our approach is quite different to that of previous acceleration methods of the tree algorithm that used GRAPE accelerators. The tree code of GRAPE also calculates the forces in a walk simultaneously; however, we cannot use the previous approach using GRAPE, and hence we redesigned the code of the host computation for the new GPU implementation.

Figure 5 shows the current parallel implementation on single computational node. A GPU board is shared by multiple processes and threads. Our code employs both process-level parallelizations using MPI and thread-level parallelizations using OpenMP. The best configuration depends on the target simulations. Multiple processes/threads perform tree traversal for different walks in parallel. Multiple processes/threads struggle for single GPU and the confliction and exclusive control deal with device driver. We experimentally choose the number of processes/threads for an optimal performance. As shown in Fig. 6, in the parallel operation, the host calculations and the GPU calculations can overlap; however, the GPU can be used continuously without a break. Thus, the effect of parallelization in the host is quite large, at least for 2 processes/threads.

Thus, we have successfully developed a novel tree algorithm that can be implemented efficiently on GPUs. Implementation results (see Fig. 1) show that our new algorithm has a speed performance that is 2.4 times than that of previous GPU implementations and 3.2 times faster than that of optimized CPU codes.

3 Real implementation on a GPU cluster

In this section, we present a real hardware implementation of a cosmological N-body simulation engine that uses our novel algorithm, on a GPU cluster. We will first describe the GPU cluster hardware before presenting real implementation results.

Figure 7 shows the basic structure of a GeForce 8800GTS GPU board (ASUS EN8800GTS TOP/512M), which is con-

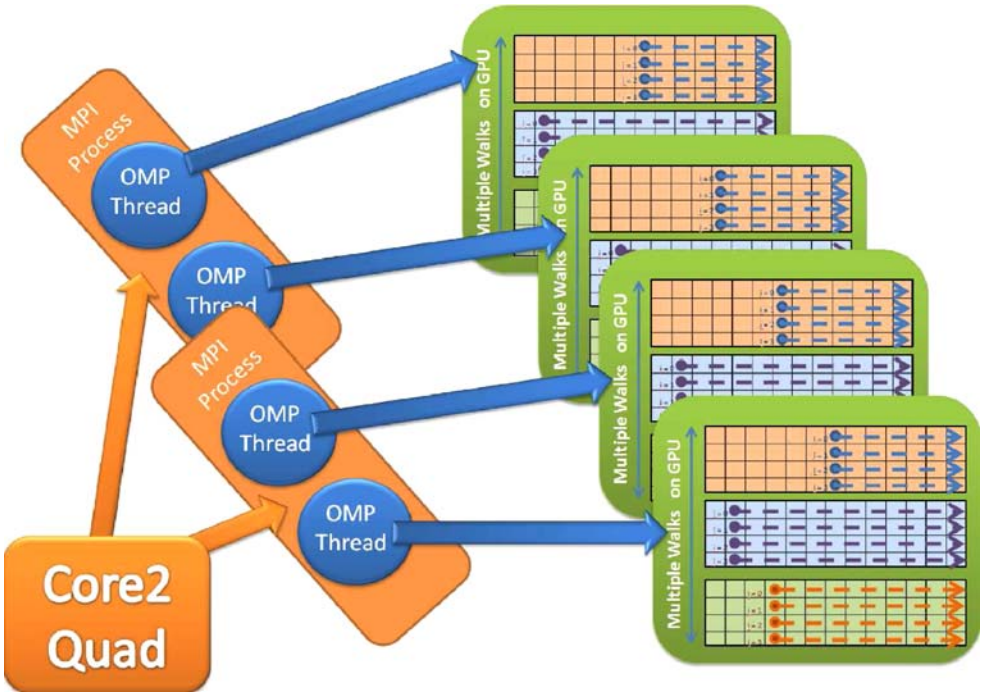

Fig. 5 Strategy for parallelization in the host program. Each process/thread treats multiple walks in parallel using a shared GPU

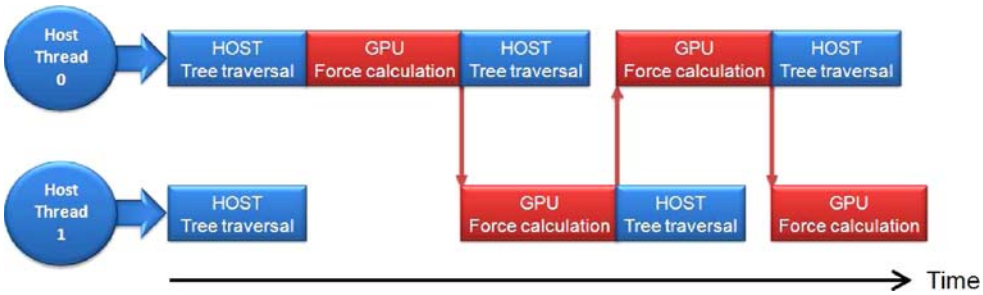

Fig. 6 Multithread calculations using a host CPU and a GPU. Overlapped calculations enable efficient use of the GPU

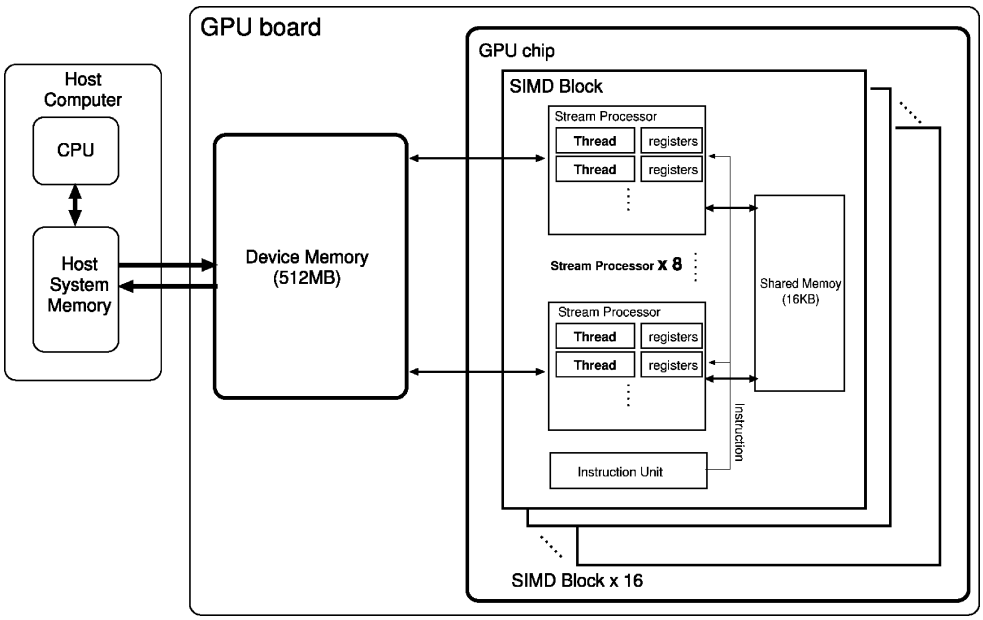

Fig. 7 Basic structure of GeForce8800GTS GPU board

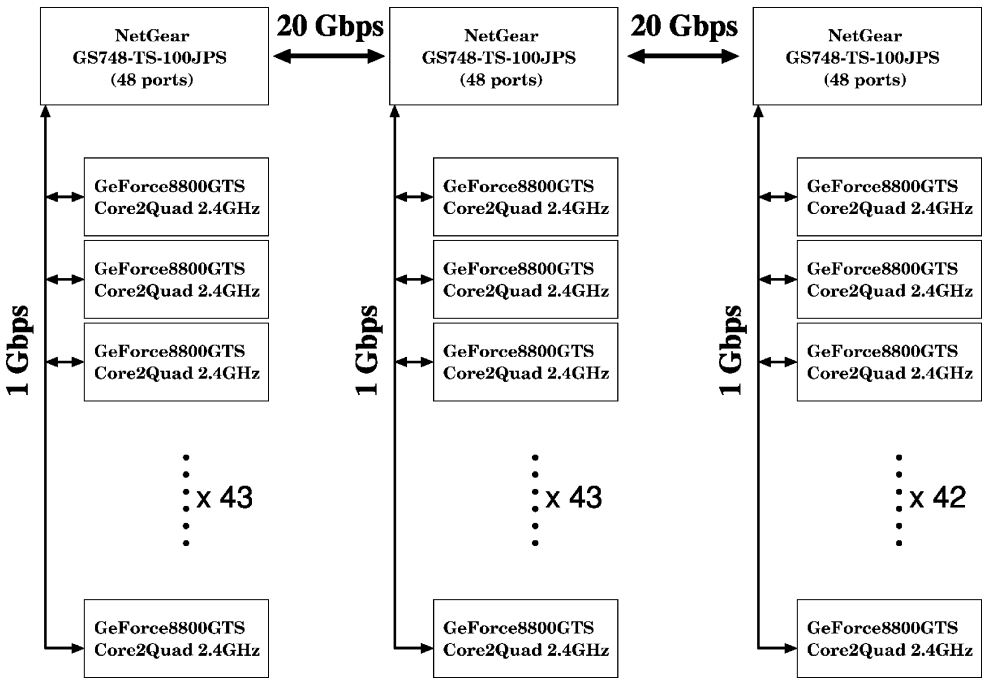

Fig. 8 Block diagram of the GPU cluster. Each node is connected through a single 1000BASE-T port to three switches, which are stacked with 20 Gbps HDMI that are linked to each other

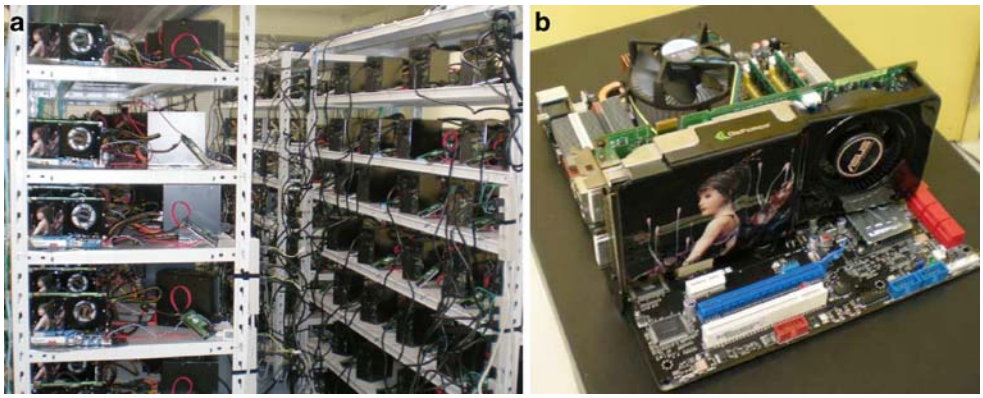

Fig. 9 Photographs of the GPU cluster (**a**) and the GPU (**b**)

nected to a host computer. The GeForce 8800GTS system is composed of a single GPU chip and a 512-MB GDDR3-DRAM (graphics double data rate 3 dynamic random access memory). The GPU chip consists of 128 stream processors (see Fig. 7).

A stream processor is a general-purpose processor, which can perform up to three single-precision floating point operations per cycle. It has a fused multiply-and-accumulate (FMA) unit and a multiply-only unit. Each stream processor works at 1.85 GHz. The peak performance of the GeForce 8800GTS is 710.4 Gflops (= 128[processors] × 3[floating-point/processor/cycle] × 1.85 [GHz]).

We now briefly describe the GPU cluster used for the real implementation reported in this paper. We used 128 host PCs with 128 GPUs for the simulation. Each PC had a CoreTM 2-Quad 2.4-GHz Q6600 processor on an X38 chipset motherboard with a single Gigabit Ethernet (GbE) port (see Fig. 8). A photograph of the GPU cluster is shown in Fig. 9.

Table 1 Price of the GPU cluster

Elements	# of elements	Price (JPY)	Price ($)
GPUs	128	6 080 000	$26 411
Host PCs	128	10 716 032	$70 429
Network switch	3	483 600	$2918
Total		17 279 632	$168 172

The costs of the constituent elements of our GPU cluster are summarized in Table 1. The prices of the GeForce 8800GTS, the host PC, and the three GbE network switches were 47 500 JPY, 83 719 JPY, and 483 600 JPY, respectively. The total price of 128 GPUs, 128 PCs, and 3 switches was 17 279 632 JPY, which is equivalent to $168 172 (102.75 JPY = $1 on April 4th 2008). All prices are inclusive of a sales tax of 5%.

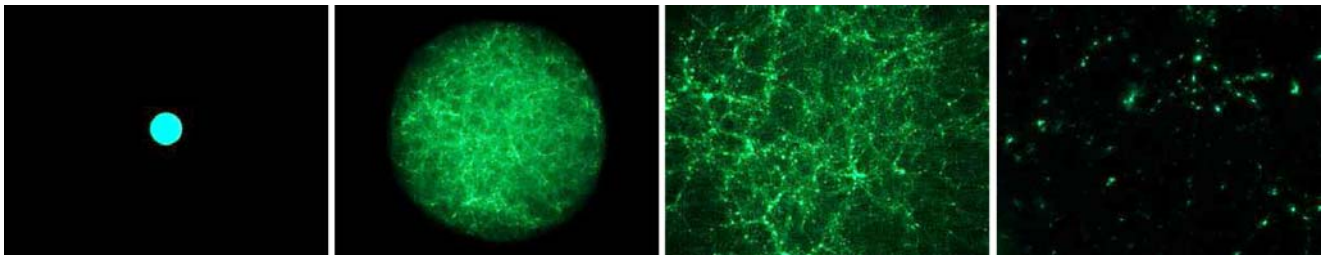

Fig. 10 Snapshots of the simulation with 8 M particles at $z = 18.8$ (*left*), $z = 4.5$ (*middle left*), $z = 2.6$ (*middle right*), and $z = 0$ (*right*)

We used CentOS for the machines that used x86_64 linux (version 5.1) as the operating system, and MPICH2 (version 1.0.7rc1) for the implementation of the message passing interface. The compilers we used for compilation of the CPU code written in the C++ language and for the compilation of our GPU code, which is also written in the C++ language, were the GNU compiler collection (version 4.1.2) and the NVIDIA CUDA compilation tools (version 1.1), respectively.

3.1 Cosmological simulation on the GPU cluster

We performed a cosmological N-body simulation of a sphere of radius 150.0 Mpc (mega parsec) with 562 232 867 particles for 394 timesteps. We assigned initial positions and velocities to particles of a spherical region selected from a discrete realization of a density contrast field based on a standard cold dark matter scenario. A particle represents 3.14×10^9 solar masses. We performed the simulation from $z = 24.0$, where z is red-shifted, to $z = 20.5$. Figure 10 shows images of the simulation with 2 M particles.

We analyzed the timing properties of the simulation for 400 timesteps. The total time of each time-step comprises:

1. the time of data transfer between the cluster nodes that use the MPI library (hereafter referred to as MPI time),
2. the time of calculation of the gravitational interactions in the GPUs (hereafter referred to as the tree-walk time). The tree-walk time also includes the time of tree traversal on the host computer,
3. and the time of miscellaneous calculations (hereafter referred to as misc time), which are not included in (1) or (2).

These timing properties are illustrated in Fig. 11. The performance data are based on the wall-clock time obtained from the system timer on the host PCs. It should be noted that the tree-walk time includes the time of data transfer between the host PC and GPU. Since the tree-walk time and the misc time are comparable, we have to accelerate the miscellaneous parts in order to improve performance.

The change in the performance with the evolution of the system is shown in Fig. 12. Here, we use an operation count of 38 operations per interaction [4, 10]. Due to the evolution of a large-scale structure in the universe, the number of interactions increased, and as a result, the performance also increased. After 50 steps the performance reached a plateau.

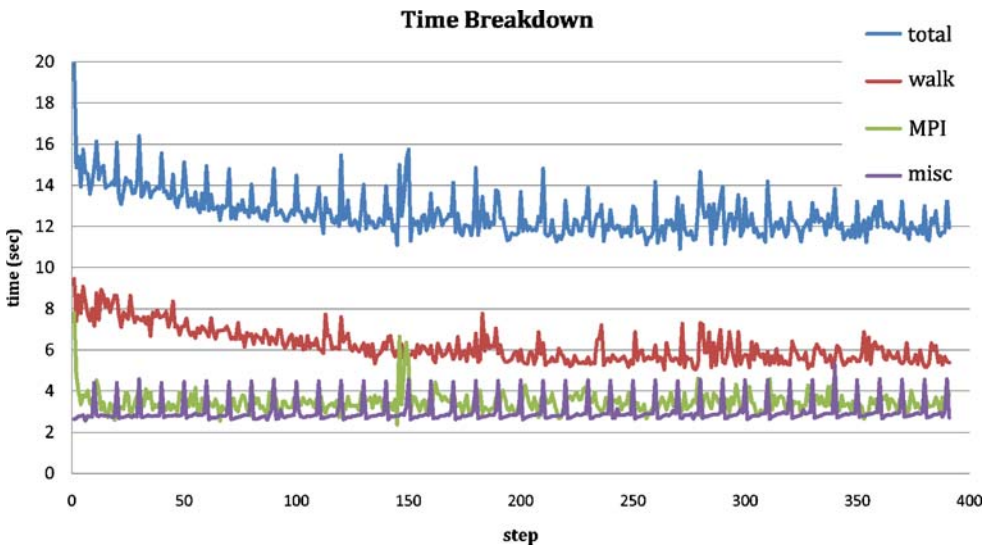

Fig. 11 Breakdown of calculation time of each time-step

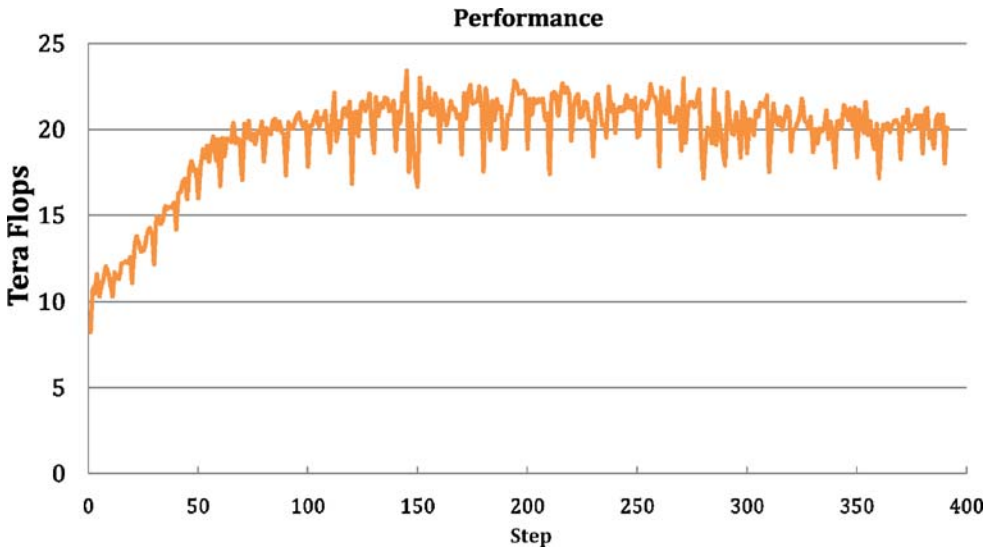

Fig. 12 Sustained performance of each timestep

In total, the number of particle-particle interactions was 2.583082×10^{15}. This implies that the average length of the interaction list was 11 661. The whole simulation took 4894 s, which included I/O operations, and the resulting average computing speed was 20.057 Gflops.

It should be noted that our modified tree algorithm performs a larger number of operations than the conventional tree algorithm that runs on a general purpose microprocessor. In order to rectify this, we estimated the operation count of the original tree algorithm for the same simulation based on the data of a simulation image (at $z = 23.9$) and the same accuracy parameter. The number of interaction counts was then found to be 1.09×10^{15}. The effective sustained speed is hence 8.499 Gflops and the resulting price/performance ratio is thus equal to a mere \$19.8 /Gflops.

Finally, it should also be noted that our modified tree algorithm is more accurate than the original tree algorithm for the same accuracy parameter, as shown in [5, 21].

4 Conclusion

GPUs can be effectively used for the high performance, and relatively low cost, implementation of N-body simulations. However, the performances of previous implementations of the simple $\mathcal{O}(N^2)$ algorithm on a GPU were not as good as that of the more practical tree algorithm, whose computation complexity is only $\mathcal{O}(N \log N)$, when run a CPU. Some variations of the tree algorithm can be used for a GPU implementation but the resulting gains in efficiency were low or inexistent compared to equivalent CPU implementations.

In this paper, we have developed a new approach to increase the effectiveness of the tree algorithm on a GPU. Our approach allows performances of approximately 200 Gflops on a GPU; which make it three times faster than that on a CPU. We performed a large cosmological N-body simulation with 562 232 867 particles on a 128 GPU cluster at a cost of \$168 000. The effective sustained speed was observed to be 8499 Gflops and the resulting price/performance was thus equal to a mere \$19.8 /Gflops. This gives a low cost implementation platform for high performance N-body simulations.

There is still considerable room for improving our GPU implementation performance however. Tree traversal on a GPU, for instance, is an important technical challenge that has a considerable impact on performance. Indeed, since it eliminates the transfer of interaction lists between the host and the GPU, the time required for CPU-GPU communication reduces considerably. A quadruple tree is also important for acceleration of the simulation, although it may decrease the performance in terms of the flops count because of its complexity. Finally, the load balancing in a GPU also has scope for improvement. For example, we can control the number of i-particles in a walk and adjust it to the block size. The implementation of these improvements will be the subject of future work.

Acknowledgement This research was supported by Special Coordination Funds for Promoting Science and Technology, Ministry of Education, Culture, Sports, Science and Technology (MEXT), Technology Integration Using Real-time Information Processing.

References

1. Barnes J, Hut P (1986) A hierarchical $O(N \log N)$ force-calculation algorithm. Nature 324:446–449
2. Warren MS, Salmon JK (1992) Astrophysical N-body simulations using hierarchical tree data structures. In: Supercomputing

'92: Proceedings of the 1992 ACM/IEEE conference on Supercomputing, pp 570–576. IEEE Computer Society Press, Los Alamitos, CA, USA
3. Fukushige T, Makino J (1996) *N*-body simulation of galaxy formation on grape-4 special-purpose computer. In: Supercomputing '96: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM), p 48. IEEE Computer Society, Washington, DC, USA. doi:http://doi.acm.org/10.1145/369028.369130
4. Warren MS, Germann TC, Lomdahl PS, Beazley DM, Salmon JK (1998) Avalon: an alpha/linux cluster achieves 10 gflops for $15 k. In: Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM), pp 1–11. IEEE Computer Society, Washington, DC, USA
5. Kawai A, Fukushige T, Makino J (1999) $7.0 /Mflops Astrophysical *N*-Body Simulation with Treecode on GRAPE-5. In: Proc of Supercomputing '99 (Gordon Bell Prize winner), pp 197–206
6. Makino J, Taiji M (1995) Astrophysical *N*-body simulations on grape-4 special-purpose computer. In: Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM), p 63. ACM, New York, NY, USA. doi:http://doi.acm.org/10.1145/224170.224400
7. Makino J, Fukushige T, Koga M (2000) A 1.349 Tflops simulation of black holes in a galactic center on grape-6. In: Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM), p 43. IEEE Computer Society, Washington, DC, USA
8. Makino J, Kokubo E, Fukushige T (2003) Performance evaluation and tuning of grape-6 – towards 40 "real" tflops. In: SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing, p 2. IEEE Computer Society, Washington, DC, USA
9. Makino J, Kokubo E, Fukushige T, Daisaka H (2002) A 29.5 Tflops simulation of planetesimals in uranus-neptune region on grape-6. In: Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing, pp 1–14. IEEE Computer Society Press, Los Alamitos, CA, USA
10. Warren MS, Salmon JK, Becker DJ, Goda MP, Sterling T (1997) Pentium Pro Inside: I. A Treecode at 430 Gflops on ASCI Red, II. Price/Performance of $50 /Mflop on Loki and Hyglac. In: Proc. Supercomputing 97, in CD-ROM. IEEE, Los Alamitos, CA
11. Springel V, White SDM, Jenkins A, Frenk CS, Yoshida N, Gao L, Navarro J, Thacker R, Croton D, Helly J, Peacock JA, Cole S, Thomas P, Couchman H, Evrard A, Colberg J, Pearce F (2005) Simulating the joint evolution of quasars, galaxies and their large-scale distribution. doi:10.1038/nature03597
12. Moore B, Diemand J, Madau P, Zemp M, Stadel J (2005) Globular clusters, satellite galaxies and stellar haloes from early dark matter peaks. doi:10.1111/j.1365-2966.2006.10116.x
13. Nyland L, Harris M, Prins J (2004) *N*-body simulations on a GPU. In: Proc of the ACM Workshop on General-Purpose Computation on Graphics Processors
14. Harris M (2005) GPGPU: General-Purpose Computation on GPUs. In: SIGGRAPH 2005 GPGPU COURSE. http://www.gpgpu.org/s2005/
15. Harris M (2005) GPGPU: General-Purpose Computation on GPUs. In: Game Developpers Conference
16. Zwart Portegies S, Belleman R, Geldof P (2007) High Performance Direct Gravitational *N*-body Simulations on Graphics Processing Unit. astro-ph/0702058
17. Hamada T, Iitaka T (2007) The chamomile scheme: An optimized algorithm for *N*-body simulations on programmable graphics processing units. http://arxiv.org/abs/astro-ph/0703100
18. Nyland L, Harris M, Prins J (2007) Fast *N*-body simulation with cuda. In: Nguyen H (ed) GPU Gems 3, chap. 31. Addison Wesley Professional
19. Belleman RG, Bedorf J, Zwart SP (2007) High performance direct gravitational *N*-body simulations on graphics processing units – ii: An implementation in cuda. doi:10.1016/j.newast.2007.07.004
20. Hamada T, Narumi T, Sakamaki T, Yasuoka K, Taiji M, Sagara T, Egami YKO (2008) The earliest scientific computation using cuda. In: Japan CUDA conference 2008, University of Tokyo
21. Barnes J (1990) A modified tree code: don't laugh; it runs. J Computat Phys 87:161–170
22. Hamada T, Ohno Y, Morimoto G, Taiji M, Toshiaki I, Nitadori K (2007) Internals of the cunbody-1 library: particle/force decomposition and reduction. Princeton, NJ
23. Makino J (2004) A Fast Parallel Treecode with GRAPE. Publ Astron Soc Japan 56(3):521–531. http://grape.astron.s.u-tokyo.ac.jp/ makino/softwares/pC++tree
24. Nitadori K, Makino J, Hut P (2006) Performance tuning of *N*-body codes on modern microprocessors: I. direct integration with a hermite scheme on x86_64 architecture. New Astron 12:169. http://www.citebase.org/abstract?id=oai:arXiv.org:astro-ph/0511062
25. Sengupta S, Harris M, Zhang Y, Owens JD (2007) Scan primitives for gpu computing. In: GH '07: Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware, pp 97–106. Eurographics Association, Aire-la-Ville, Switzerland